Datalogisk Institut
KØBENHAVNS UNIVERSITET

Computing the Maximum Detour of a Plane Graph in Subquadratic Time

Christian Wulff-Nilsen

Technical Report no. 08-07
ISSN: 0107-8283

Dept. of Computer Science
University of Copenhagen • Universitetsparken 1
DK-2100 Copenhagen • Denmark

Computing the Maximum Detour of a Plane Graph in Subquadratic Time

Christian Wulff-Nilsen *

June 25 2008

Abstract

Let G be a plane graph where each edge is a line segment. We consider the problem of computing the maximum detour of G, defined as the maximum over all pairs of distinct points p and q of G of the ratio between the distance between p and q in G and the distance $|pq|$. The fastest known algorithm for this problem has $\Theta(n^2)$ running time where n is the number of vertices. We show how to obtain $O(n^{3/2}\log^3 n)$ expected running time. We also show that if G has bounded treewidth, its maximum detour can be computed in $O(n\log^3 n)$ expected time.

1 Introduction

Given a geometric graph G, its stretch factor (or dilation) is the maximum over all pairs of distinct vertices u and v of the ratio between the distance between u and v in G and the Euclidean distance $|uv|$ between u and v.

A spanner is a network with small stretch factor. Spanners that keep other cost measures low, such as size, weight, degree, and diameter, are important structures in areas such as VLSI design, distributed computing, and robotics. Proving the existence of spanners that simultaneously keep several cost measures low and efficiently computing such spanners for a given

*Department of Computer Science, University of Copenhagen, koolooz@diku.dk, http://www.diku.dk/~koolooz/

set of vertices is an active area of research [2, 9]. For more on spanners, see surveys [6, 12], and the book by Narasimhan and Smid [11].

An interesting dual problem is the following: given a geometric graph, compute its stretch factor. A related problem which we consider in this paper is that of computing the maximum detour of a plane graph, which is defined like stretch factor except that the maximum is taken over all pairs of distinct *points* of the graph, not just the vertices.

If the graph is planar then its stretch factor can be computed in $\Theta(n^2)$ time, where n is the number of vertices, by applying the APSP algorithm by Frederickson [7]. This bound also holds for the problem of computing the maximum detour of a plane graph [1]. It is an open problem whether subquadratic time algorithms exist.

For more special types of graphs, faster algorithms are known. If the graph is a path, its stretch factor and maximum detour can be computed in $O(n \log n)$ expected time and if the graph is a tree or a cycle, its stretch factor and maximum detour can be computed in $O(n \log^2 n)$ expected time [1]. Using parametric search gives $O(n \operatorname{polylog} n)$ worst-case time algorithms for these special types of graphs. Also, if the graph has bounded treewidth k, its stretch factor can be computed in $O(n \log^{k+1} n)$ expected time [3].

In this paper, we show how to compute the maximum detour of a plane graph with n vertices in $O(n^{3/2} \log^3 n)$ expected time, thereby solving the open problem of whether a subquadratic time algorithm exists for this problem. We also show that if the graph has bounded treewidth, its maximum detour can be computed in $O(n \log^3 n)$ expected time.

The organization of the paper is as follows. In Section 2, we give various definitions and introduce some notation. In Section 3, we make use of the separator theorem by Lipton and Tarjan which enables us to apply the divide-and-conquer paradigm to the input graph. We define colourings of points of a face of the graph in Section 4, show some properties of these colourings and how to efficiently compute them. In Sections 5 and 6, we show how the colourings give an efficient way of computing the maximum detour between points on a face of the graph and this in turn gives an efficient algorithm for computing the maximum detour of the entire graph. In Section 7, we apply the same ideas to plane graphs with bounded treewidth. Finally, we make some concluding remarks in Section 8.

2 Definitions and Notation

Let $G = (V, E)$ be a plane graph where each edge is a line segment and let P_G be the set of points of G (vertices as well as interior points of edges). Given two points $p, q \in P_G$, we define $d_G(p, q)$ as the length of a shortest path in G between p and q, where the length of a path is measured as the sum of the Euclidean lengths of the (parts of) edges on this path. If there is no such path, we define $d_G(p, q) = \infty$. If $p \neq q$ then the *detour* $\delta_G(p, q)$ between p and q (in G) is defined as the ratio $d_G(p, q)/|pq|$. The *maximum detour* δ_G of G is the maximum of this ratio over all pairs of distinct points of P_G.

Where appropriate, we will regard a plane graph as the set of points belonging to the graph. So for instance, if G and H are plane graphs then $G \cap H$ is the set of points belonging to both G and H. If well-defined, we will regard the resulting point set as a graph.

For a graph G, we let $|G|$ denote its size, i.e. the number of vertices plus the number of edges in G. Given two subsets P_1 and P_2 of the set P_G of points of G, we define

$$\delta_G(P_1, P_2) = \max_{p \in P_1, q \in P_2, p \neq q} \delta_G(p, q)$$

If p is a point of G and $P \subseteq P_G$, we write $\delta_G(p, P) = \delta_G(P, p)$ instead of $\delta_G(\{p\}, P)$ and we write $\delta_G(P)$ as a shorthand for $\delta_G(P, P)$. We extend these definitions to subgraphs, edges, and vertices by regarding them as sets of points.

Given paths $P = p_1 \to p_2 \to \ldots \to p_r$ and $Q = q_1 \to q_2 \to \ldots \to q_s$, where $p_r = q_1$, we let PQ denote the combined path $p_1 \to p_2 \to \ldots \to p_{r-1} \to q_1 \to q_2 \to \ldots \to q_s$. For a vertex v, we let $v \to P$ denote the path $v \to p_1 \to p_2 \to \ldots \to p_r$.

3 Separating the Problem

In all the following, let $G = (V, E)$ be an n-vertex plane graph in which edges are line segments. We seek to compute δ_G in $O(n^{3/2} \log^3 n)$ expected time. We will assume that G is connected since otherwise, the problem is trivial.

To compute δ_G, we apply the divide-and-conquer paradigm. In this section, we separate G into two smaller graphs, G_A and G_B, of roughly the same

size. We recursively compute $\delta_G(G_A)$ and $\delta_G(G_B)$ and in Section 5, we show how to efficiently obtain the maximum detour $\delta_G(G_A, G_B)$.

To separate our problem, we use the separator theorem of Lipton and Tarjan [10]. This gives us, in $O(n)$ time, a partition of V into three sets, A, B, and P, such that

1. no edge joins a vertex in A with a vertex in B,
2. neither A nor B contains more than $n/2$ vertices, and
3. P contains no more than $\frac{2\sqrt{2}}{1-\sqrt{2/3}}\sqrt{n}$ vertices.

We refer to vertices of P as *portals*. In all the following, we let k denote the number of portals and let $p_1, \ldots, p_k$ denote the portals.

Having found this partition, we compute and store shortest path lengths from each portal to each vertex of V. This can be done in $O(n^{3/2} \log n)$ time using Dijkstra's SSSP algorithm for each portal.

Let G_A be the subgraph of G induced by $A \cup P$ and let G_B be the subgraph of G induced by $B \cup P$. We construct G_A and G_B and recursively compute $\delta_G(G_A)$ and $\delta_G(G_B)$. Clearly,

$$\delta_G = \max\{\delta_G(G_A), \delta_G(G_B), \delta_G(G_A, G_B)\}.$$

In the following, we deal with the problem of computing $\delta_G(G_A, G_B)$ and in Section 6, we show how to recursively compute $\delta_G(G_A)$ and $\delta_G(G_B)$.

We will need the following lemma which is a generalization of a result in [5] (we omit the proof since it is virtually identical to that in [5]).

Lemma 1. *Maximum detour δ_G is achieved by a pair of co-visible points.*

This result allows us to consider only detours between pairs of points of the same face of G. In all the following, let f be a face of G, let $f_A = f \cap G_A$, and let $f_B = f \cap G_B$. We will show how to compute $\delta_G(f_A, f_B)$ in $O(|f|k \log^2 n)$ expected time. From this it will follow that $\delta_G(G_A, G_B)$ can be found in $O(n^{3/2} \log^2 n)$ expected time.

We will assume that f is an internal face of G. The external face is dealt with in a similar way. We also assume that f_A and f_B do not share any edges since any edge e shared by them must have portals as both endpoints and the detours from points in e to all points in G will be considered in the two

recursive calls that compute $\delta_G(G_A)$ and $\delta_G(G_B)$, respectively. Hence, we may disregard e when computing $\delta_G(f_A, f_B)$.

Assume for now that f is simple. The case where f is non-simple is considered in Section 5.3.

4 Colouring Points

We now define colourings of points of f. As we shall see in Section 5, these colourings will prove helpful when computing $\delta_G(f_A, f_B)$.

Face f is defined by a simple cycle $v_1 \to v_2 \to \ldots \to v_{n_f} \to v_1$ such that the interior of f is to the left as we walk in the direction specified by the vertices.

For a point $p \in f$, we let $d_f(p)$ denote the Euclidean length of the path from v_1 to p that visits vertices in the order specified above. We define an order $<^f_{v_1}$ of the set of points of f as follows. For two points p and q of f, $p <^f_{v_1} q$ if and only if $d_f(p) < d_f(q)$. Order $p >^f_{v_1} q$ is defined in a similar way. In the following, we assume that the sum of lengths of all edges in f and $d_f(v_i)$ for all $v_i \in f$ are precomputed.

By starting the walk in any other vertex v_i of f, we can similarly define orders $<^f_{v_i}$ and $>^f_{v_i}$. It is easy to see that determining whether e.g. $p <^f_{v_i} q$ holds for any points $p, q \in f$ can be done in constant time using the above precomputed values. Where appropriate, we will regard the points of f (or f_A or f_B) occuring between two points w.r.t. order $<^f_{v_i}$ as an interval of points.

In the following, let a be a vertex in f_A and let $a' \in f \cap V$ be its successor w.r.t. $<^f_a$. We now consider associating with a a colouring of points in f_B using colours $c_1, \ldots, c_k$. A point $p \in f_B$ is given colour c_i if portal p_i is on a shortest path in G from a to p. In case of ties, pick the colour such that the corresponding portal has minimum distance to a in G. In case of further ties, pick the colour with the smaller index. We let $c_a(p)$ denote the colour assigned to p.

We will show that colours occur in intervals as we walk around f_B with each colour assigned to at most one interval. Furthermore, we will show that the order of these intervals is induced by an order of the portals which we define next.

Let u_0 and u_1 be distinct vertices of G connected by an edge and consider a portal p_i. Choose edge $(u_1, u_2) \in E$ such that u_2 is on a shortest path from u_1 to p_i and such that $u_0 \to u_1 \to u_2$ makes the sharpest possible left turn

at u_1 (if a left turn is not possible we regard the least possible right turn as a sharpest possible left turn and we regard a turn of angle π as a left turn of angle π).

Repeat this procedure by picking, for $j = 3, \ldots, r$, an edge (u_{j-1}, u_j) such that u_j is on a shortest path from u_{j-1} to p_i and such that $u_{j-2} \to u_{j-1} \to u_j$ makes the sharpest possible left turn at u_{j-1}; here, r is the smallest index such that $u_j = p_i$. The resulting path $u_1 \to u_2 \to \ldots \to u_r$ is uniquely defined and is a shortest path from u_1 to p_i. We denote it by $\overleftarrow{P_i}(u_0, u_1)$. We define $\overleftarrow{P_i}'(u_0, u_1) = u_0 \to \overleftarrow{P_i}(u_0, u_1)$. In the following, we will write $\overleftarrow{P_i}$ resp. $\overleftarrow{P_i'}$ as a shorthand for $\overleftarrow{P_i}(a', a)$ resp. $\overleftarrow{P_i'}(a', a)$, where a and a' are defined as above.

For two distinct portals p_i and p_j we write $p_i \prec_a^f p_j$ if $p_i \in \overleftarrow{P_j}$ or if $\overleftarrow{P_i'}$ makes a sharper left turn than $\overleftarrow{P_j'}$ at some shared interior vertex.

Lemma 2. *If paths $\overleftarrow{P_i}$ and $\overleftarrow{P_j}$ split at a vertex v they cannot meet after v.*

Proof. Suppose the lemma does not hold. Let w be a vertex where $\overleftarrow{P_i}$ and $\overleftarrow{P_j}$ meet after v. Assume w.l.o.g. that $\overleftarrow{P_i}$ makes a sharper left turn at v than $\overleftarrow{P_j}$. Replacing the subpath of $\overleftarrow{P_j}$ from v to w by the subpath of $\overleftarrow{P_i}$ from v to w gives a shortest path from a to p_j but this contradicts the assumption that $\overleftarrow{P_j'}$ makes the sharpest possible left turn at v. □

Lemma 3. *The relation $\prec_a^f$ above is a strict total order of the portals.*

Proof. Let p_{i_1}, p_{i_2}, and p_{i_3} be three distinct portals. Clearly, $p_{i_1} \prec_a^f p_{i_2}$ or $p_{i_2} \prec_a^f p_{i_1}$, showing that the relation $\prec_a^f$ is total.

We need to show that $p_{i_1} \prec_a^f p_{i_2}$ and $p_{i_2} \prec_a^f p_{i_1}$ cannot both hold and we need to show transitivity: if $p_{i_1} \prec_a^f p_{i_2}$ and $p_{i_2} \prec_a^f p_{i_3}$ then $p_{i_1} \prec_a^f p_{i_3}$.

To show the first part, suppose for the sake of contradiction that $p_{i_1} \prec_a^f p_{i_2}$ and $p_{i_2} \prec_a^f p_{i_1}$. Then we cannot have $p_{i_1} \in \overleftarrow{P_{i_2}}$ and we cannot have $p_{i_2} \in \overleftarrow{P_{i_1}}$. Consider the first vertex v at which paths $\overleftarrow{P_{i_1}}$ and $\overleftarrow{P_{i_2}}$ split. Then they cannot meet after v by Lemma 2. But this implies that one of the two paths $\overleftarrow{P_{i_1'}}$ and $\overleftarrow{P_{i_2'}}$ cannot make a sharper left turn than the other path at any shared interior vertex, a contradiction.

To show transitivity, suppose $p_{i_1} \prec_a^f p_{i_2}$ and $p_{i_2} \prec_a^f p_{i_3}$. We need to show that either $p_{i_1} \in \overleftarrow{P_{i_3}}$ or that $\overleftarrow{P_{i_1}}'$ makes a sharper left turn than $\overleftarrow{P}'_{i_3}$ at some interior vertex of $\overleftarrow{P}'_{i_3}$.

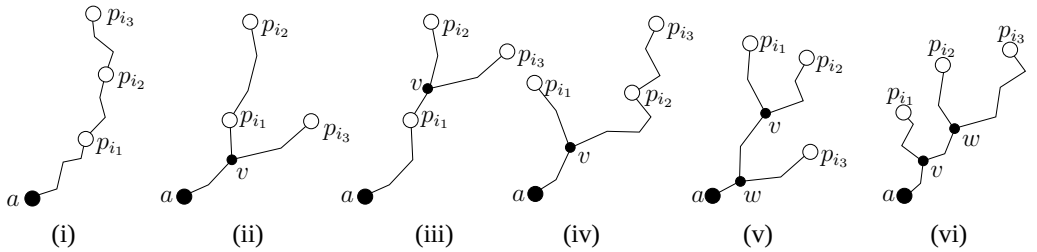


Figure 1: The six cases considered in Lemma 3.

Assume first that $p_{i_1} \in \overleftarrow{P_{i_2}}$. If $p_{i_2} \in \overleftarrow{P_{i_3}}$ then $p_{i_1} \in \overleftarrow{P_{i_3}}$ (Figure 1(i)). If $p_{i_2} \notin \overleftarrow{P_{i_3}}$ then $\overleftarrow{P_{i_2}}'$ makes a sharper left turn than $\overleftarrow{P_{i_3}}'$ at some interior vertex v of $\overleftarrow{P_{i_3}}'$. If $v \in \overleftarrow{P_{i_1}} \setminus \{p_{i_1}\}$ (Figure 1(ii)) then $\overleftarrow{P_{i_1}}'$ makes a sharper left turn than $\overleftarrow{P_{i_3}}'$ at v. And if $v \notin \overleftarrow{P_{i_1}} \setminus \{p_{i_1}\}$ (Figure 1(iii)) then $p_{i_1} \in \overleftarrow{P_{i_3}}$. In all cases, $p_{i_1} \prec_a^f p_{i_3}$.

Now, assume that $p_{i_1} \notin \overleftarrow{P_{i_2}}$. Then $\overleftarrow{P_{i_1}}'$ makes a sharper left turn than $\overleftarrow{P_{i_2}}'$ at some interior vertex v of $\overleftarrow{P_{i_2}}'$. If $p_{i_2} \in \overleftarrow{P_{i_3}}$(Figure 1(iv)) then $\overleftarrow{P_{i_1}}'$ makes a sharper left turn than $\overleftarrow{P_{i_3}}'$ at v. If $p_{i_2} \notin \overleftarrow{P_{i_3}}$ then let w be a vertex such that $\overleftarrow{P_{i_2}}'$ makes a sharper left turn than $\overleftarrow{P_{i_3}}'$ at w. If $w \in \overleftarrow{P_{i_1}}$ (Figure 1(v)) then $\overleftarrow{P_{i_1}}'$ makes a sharper left turn than $\overleftarrow{P_{i_3}}'$ at w. And if $w \notin \overleftarrow{P_{i_1}}$ (Figure 1(vi)) then $\overleftarrow{P_{i_1}}'$ makes a sharper left turn than $\overleftarrow{P_{i_3}}'$ at v. So again, $p_{i_1} \prec_a^f p_{i_3}$. □

We now show the relation between the order $\prec_a^f$ of portals and the order $<_a^f$ of vertices in f_B.

Theorem 1. *Let p and q be two distinct points of f_B and assume that $c_a(p) = c_i$ and $c_a(q) = c_j$, $i \neq j$. Then $p_i \prec_a^f p_j$ if and only if $p <_a^f q$.*

Proof. By symmetry, it is enough to show that if $p_i \prec_a^f p_j$ then $p <_a^f q$.

So assume that $p_i \prec_a^f p_j$. Since $c_a(q) = c_j$, we have $p_i \notin \overleftarrow{P_j}$. It follows that there is a vertex v at which $\overleftarrow{P_i}$ and $\overleftarrow{P_j}$ split and $\overleftarrow{P_i'}$ makes a sharper left turn at v than $\overleftarrow{P_j}'$. By Lemma 2, the two paths do not meet again after v. In particular, $\overleftarrow{P_i}$ does not cross $\overleftarrow{P_j}$, see Figure 2.

Let P_p be a shortest path from p_i to p. Then P_p cannot intersect $\overleftarrow{P_j}$ since then p_i and p_j would both be on a shortest path from a to q with $d_G(a, p_i) < d_G(a, p_j)$, contradicting the assumption that $c_a(q) = c_j$.

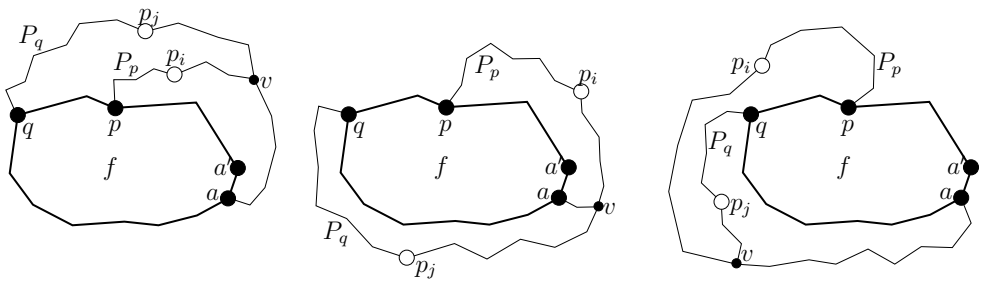


Figure 2: The possible situations in the proof of Theorem 1 when $p_i \prec_a^f p_j$.

Let P_q be a shortest path from p_j to q. Then $\overleftarrow{P_i}$ cannot intersect P_q since otherwise, p_i and p_j would both be on a shortest path from a to p with $d_G(a, p_j) < d_G(a, p_i)$, contradicting the assumption that $c_a(p) = c_i$.

Furthermore, P_q cannot intersect P_p. For assume it did. Then there would be a shortest path from a to p through p_j and a shortest path from a to q through p_i. If $d_G(a, p_i) < d_G(a, p_j)$ then $c_a(q) \neq c_j$ and if $d_G(a, p_j) < d_G(a, p_i)$ then $c_a(p) \neq c_i$. Hence, $d_G(a, p_i) = d_G(a, p_j)$. But then $i < j$ would imply $c_a(q) \neq c_j$ and $i > j$ would imply $c_a(p) \neq c_i$. This contradicts the colours assigned to p and q.

It follows from the above that paths $\overleftarrow{P_i}P_p$ and $\overleftarrow{P_j}P_q$ do not intersect except in the vertices they share until reaching v, see Figure 2. This implies that $p <_a^f q$, showing the theorem. □

Corollary 1. *Interval f_B can be split up into $O(k)$ sub-intervals such that points in the same sub-interval are assigned the same colour w.r.t. vertex $a \in f_A$.*

If the sub-intervals of Corollary 1 are picked such that they have maximal size, we refer to them as *colour intervals of a*.

We now show how the colour intervals of a can be computed efficiently, after some preprocessing.

Lemma 4. *The order of colour intervals of a can be computed in $O(k \log^2 n)$ time assuming $O(kn \log n)$ time for preprocessing. The preprocessing step is independent of a and f.*

Proof. We will prove the lemma by presenting a data structure that, given distinct portals p_i and p_j, determines whether $p_i \prec_a^f p_j$ and does so for any

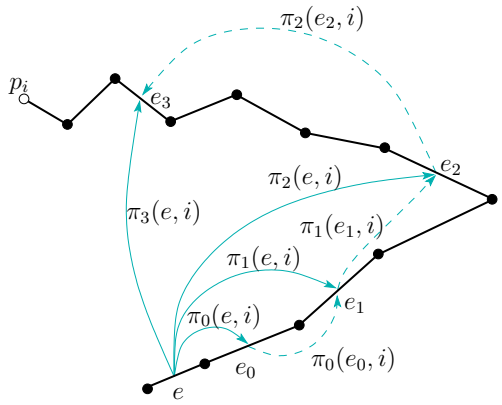


Figure 3: Example with π-pointers from edge e to edges $e_0 = \pi_0(e, i)$, $e_1 = \pi_1(e, i)$, $e_2 = \pi_2(e, i)$, and $e_3 = \pi_3(e, i)$ on $\overleftarrow{P_i}(e)$. Note that $e_1 = \pi_0(\pi_0(e, i), i)$, $e_2 = \pi_1(\pi_1(e, i), i)$, and $e_3 = \pi_2(\pi_2(e, i), i)$. Here, $j_e = 3$.

a and f. We will show how to construct this data structure in $O(kn \log n)$ time such that it can determine whether $p_i \prec_a^f p_j$ in $O(\log n)$ time. This will allow us to sort the portals according to $\prec_a^f$ in time $O(k \log^2 n)$ time using a sorting algorithm like merge or heap sort since such an algorithm performs $O(k \log k)$ comparisons. This result together with Theorem 1 will show the lemma.

We first show how to construct the data structure. In the following, let E' be the set of directed edges obtained by regarding each edge of E as two oppositely directed edges.

With each edge $e = (u, v) \in E'$ and each portal p_i, we associate pointers $\pi_j(e, i)$, $j = 0, \ldots, j_e$. Pointer $\pi_j(e, i)$ points to the edge e_j of $\overleftarrow{P_i}(e)$ such that the number of edges between e and e_j in $\overleftarrow{P_i}(e)$ is $2^j - 1$, see Figure 3. Here, j_e is the largest j such that e_j exists. Note that $j_e = O(\log n)$ so the total number of pointers over all edges e and all portals p_i is $O(kn \log n)$.

Suppose we are given these pointers. Then for any edge e and any portals p_i and p_j, binary search allows us to determine, in $O(\log n)$ time, the vertex at which $\overleftarrow{P_i}(e)$ and $\overleftarrow{P_j}(e)$ split. If they do not split, binary search will also detect this and determine whether $p_i \in \overleftarrow{P_j}(e)$ or $p_j \in \overleftarrow{P_i}(e)$. From this it follows that we can determine whether $p_i \prec_a^f p_j$ in $O(\log n)$ time, given these pointers.

We now show how to construct the pointers in $O(kn \log n)$ time. More

specifically, given a portal p_i, we show how to compute pointers $\pi_j(e,i)$, $e \in E$, $j = 1, \ldots, j_e$, in a total of $O(n \log n)$ time.

For each edge (u,v) of E', we colour it white if a shortest path from u to p_i goes through v, that is, if $d_G(u,p_i) = |uv| + d_G(v,p_i)$. Otherwise, we colour it black.

Now, for each vertex v of G, we consider the edges of E' starting or ending in v in counter-clockwise order. For each edge (u,v) ending in v, we add a pointer to the previous (in the counter-clockwise order) white edge (v,w) that starts in v. Note that (v,w) is the edge satisfying that $u \rightarrow v \rightarrow w$ makes the sharpest possible left turn such that (v,w) is on a shortest path in G from v to p_i.

Next, we consider the edges of E' in some order $e_1, \ldots, e_{|E'|}$. For edge $e_1 = (u_1, v_1)$, we compute $\overleftarrow{P_i}(e_1)$ by starting in v_1 and then computing subsequent vertices using the pointers just added. We colour e_1 red and as we visit the edges of $\overleftarrow{P_i}(e_1)$, we colour them red as well.

We then compute pointers $\pi_j(e,i)$ for each edge $e \in \overleftarrow{P'_i}(e_1)$ and for each $j = 0, \ldots, j_e$. Below, we show how to do this efficiently.

For edge e_j, $j > 1$, we do exactly the same as for e_1 except that we stop when reaching a red edge. All visited edges including e_j are coloured red and π-pointers are computed for each of these edges.

The above algorithm is correct since it computes π-pointers for all edges. Its running time, excluding the time to compute π-pointers, is $O(n \log n)$. To see this, note that sorting edges counter-clockwise for each vertex takes a total of $O(n \log n)$ time. Colouring edges black and white takes $O(n)$ time. Finally, visiting (parts of) paths $\overleftarrow{P_i}(e_j)$ takes time proportional to the number of edges coloured red which is $O(n)$.

What remains is to show how to compute π-pointers in $O(n \log n)$ time. We observe that $\pi_j(e,i)$, $j = 0, \ldots, j_e$, can be computed in a total of $O(\log n)$ time if π-pointers of all edges of $\overleftarrow{P_i}(e)$ have been computed. This follows easily from repeated applications of the identity $\pi_j(e,i) = \pi_{j-1}(\pi_{j-1}(e,i),i)$, $j > 0$ (see Figure 3). Hence, by computing π-pointers in the opposite order of the order in which edges are coloured red, we obtain all π-pointers in $O(n \log n)$ time. □

Theorem 2. *Endpoints of colour intervals of a can be computed in $O(k \log^2 n)$ time assuming $O(kn \log n)$ preprocessing time. The preprocessing step is independent of a and f.*

Proof. We start by computing the order of colour intervals of a. By Lemma 4, this can be done in $O(k \log^2 n)$ time with $O(kn \log n)$ preprocessing time. Let π be the permutation of $\{1, \ldots, k\}$ such that $p_{\pi(1)} \prec_a^f \ldots \prec_a^f p_{\pi(k)}$.

We then compute the colour of the first point $b_{\min}$ and the last point $b_{\max}$ of f_B w.r.t. the order $<_a^f$. This can be done in time proportional to the number k of colours since SSSP lengths for each portal have been precomputed.

If $c_a(b_{\min}) = c_a(b_{\max})$ then by Theorem 1, all points between $b_{\min}$ and $b_{\max}$ have this colour. In this case, the algorithm associates this colour with the sub-interval between the two vertices and returns (the sub-interval is not stored explicitly, only its end vertices $b_{\min}$ and $b_{\max}$).

Otherwise, a vertex $b \in f_B$ is picked, such that the number of edges in f_B before resp. after b w.r.t. the order $<_a^f$ is (approximately) the same, and its colour $c_a(b)$ is computed. Let i be the index such that $c_{\pi(i)} = c_a(b)$. The algorithm calls itself recursively on vertices between $b_{\min}$ and b with colours $c_{\pi(1)}, \ldots, c_{\pi(i)}$. And it calls itself recursively on vertices between b and $b_{\max}$ with colours $c_{\pi(i)}, \ldots, c_{\pi(k)}$.

The recursion stops when $b_{\min}$ and $b_{\max}$ are the endpoints of a single edge e of f_B. If $c_a(b_{\min}) = c_a(b_{\max})$, the algorithm associates this colour with e and returns. Otherwise, we need the following simple observation: there is a point p on e such that all points on $b_{\min}p$ have colour $c_a(b_{\min})$ and all points on $pb_{\max}$ have colour $c_a(b_{\max})$. Furthermore, p can be computed in $O(1)$ time, given these two colours. The algorithm associates the two colours with their respective segments of e and returns.

When the algorithm terminates, each colour c_i is associated with $O(\log n)$ sub-intervals and their union defines the colour interval of a with colour c_i. Finding the colour intervals of a from these $O(k \log n)$ sub-intervals takes $O(k \log n)$ time. What remains is to show that the algorithm above has $O(k \log^2 n)$ running time.

Let $T(m, k)$ be a function expressing the time for the above algorithm where m is the number of edges and k is the number of colours. If we assume that vertices of f_B are stored in an array then the point b that splits points of f_B into two equal halves can be found in $O(1)$ time. Thus, there is a constant $c' > 0$ such that the algorithm uses at most $c'k$ time steps excluding time spent in recursive calls. There is also a constant c'' such that $T(m, k) \leq c''k$ when $m \leq 2$. Let $c = \max\{c', c''\}$. Then (ignoring floors and ceilings)

$$T(m, k) \leq ck + T(m/2, k_1) + T(m/2, k_2),$$

where $m > 2$ and $k_1, k_2 \in \{1, \ldots, k\}$, $k_1 + k_2 = k + 1$. Let $\tilde{T}(m, k)$ be the

running time $T(m,k)$ minus a value of $c\log n$ charged to each split vertex b encountered in the current and in recursive calls. We claim that $\tilde{T}(m,k) \leq ck\log m$.

The proof is by induction on $m \geq 2$. When $m = 2$, we spend at most $c''k \leq ck$ time. Now, assume that $m > 2$. The induction hypothesis gives

$$\begin{aligned}\tilde{T}(m,k) &\leq ck + \tilde{T}(m/2,k_1) + \tilde{T}(m/2,k_2) - c\log n\\ &\leq ck + c(k+1)\log(m/2) - c\log n\\ &= ck\log m + c\log m - c\log n - c\\ &< ck\log m,\end{aligned}$$

as requested.

It follows that $T(m,k) \leq ck\log m + xc\log n$, where x is the total number of split vertices. This number is proportional to the number of sub-intervals returned by the algorithm which is $O(k\log m)$. This shows the theorem. □

Note that a colour interval I need not be closed, i.e. one or both of the endpoints need not belong to I.

We conclude this section with the following simple result which will prove useful when we compute detours between points that may be interior points of edges.

Lemma 5. *Let p a point of edge $e = (u,v)$ of G and let q be a point in G. Suppose that p_i is on a shortest path from u to q and that p_j is on a shortest path from v to q. Then either p_i or p_j is on a shortest path from p to q.*

Proof. A shortest path from p to q goes through either u or v. □

5 The Detour of Points in a Face

In this section, we show how to compute $\delta_G(f_A, f_B)$ in $O(|f|k\log^2 n)$ expected time.

We start by computing, for each edge $e = (u,v) \in f_A$, $O(k)$ colour intervals of u and of v using $O(k\log^2 n)$ time (with $O(kn\log n)$ preprocessing) and take the union of the endpoints of these colour intervals. This gives $O(k)$ smaller sub-intervals which we associate with e. The total running time for this over all edges is $O(|f|k\log^2 n)$.

Now, let P be one of the sub-intervals associated with edge $e = (u,v)$. Then there are $i,j \in \{1,\ldots,k\}$ such that $c_u(p) = c_i$ and $c_v(p) = c_j$ for all

$p \in P$. Hence, for any point $q \in P$, p_i is on a shortest path from u to q and p_j is on a shortest path from v to q. Lemma 5 implies that for any point $p \in e$ and any point $q \in P$, either p_i or p_j is on a shortest path from p to q.

We refer to P as a *type* 1-*interval* (of e) if $c_i \neq c_j$ and a *type* 2-*interval* (of e) if $c_i = c_j$.

For any edge e of f_A and any type i-interval P of e, $i = 1, 2$, we may assume that P is a closed interval having endpoints in vertices of f_B. For otherwise, we could compute the maximum detour between e and the first resp. last edge e' of P (all other edges of P have endpoints in vertices of f_B). Computing $\delta_G(e, e')$ is a constant-size problem (when SSSP lengths for each portal have been precomputed) since we know that for each point $p \in e$ and each point $p' \in e$, there is a shortest path from p to p' through either of two portals p_i and p_j. Thus, it takes $O(1)$ time to compute $\delta_G(e, e')$ (see also [1]). Over all e and P, this amounts to $O(|f|k)$ time.

The value $\delta_G(f_A, f_B)$ is computed in two phases. In phase i, the maximum detour between points in edges of f_A and points in associated type i-intervals are computed, $i = 1, 2$.

5.1 Phase 1

We will show that phase 1 takes $O(|f|k)$ time when shortest path lengths from portals and colour intervals have been computed.

The algorithm for this phase is straightforward. For each edge e of f_A it considers all edges e' of each type 1-interval of e and computes $\delta_G(e, e')$ in constant time.

To show that phase 1 takes $O(f|k|)$ time, we need to show that the number of edge pairs (e, e') is $O(f|k|)$. To do this, we introduce a so called *dual colouring* of vertices of f_A for each vertex of f_B. Let b be a vertex of f_B. Then vertex $a \in f_A$ is given *dual colour* $c_b(a)$, defined as the colour $c_a(b)$.

Assigning dual colours to all vertices of f_A partitions this set into maximal sub-intervals with vertices in each sub-interval having the same dual colour. We refer to these sub-intervals as the *dual colour intervals of* b. These dual colour intervals will help us bound the number of edge pairs (e, e'). First, we need two lemmas.

Lemma 6. *Let b be a vertex of f_B and let a, a_1, a_2 be vertices of f_A such that $a_1 <_b^f a <_b^f a_2$, $c_b(a_1) = c_b(a_2) = c_i$, and $c_b(a) = c_j$, $i \neq j$. Then for any vertex a' of f_A, $c_b(a') \neq c_j$ if either $a' <_b^f a_1$ or $a' >_b^f a_2$.*

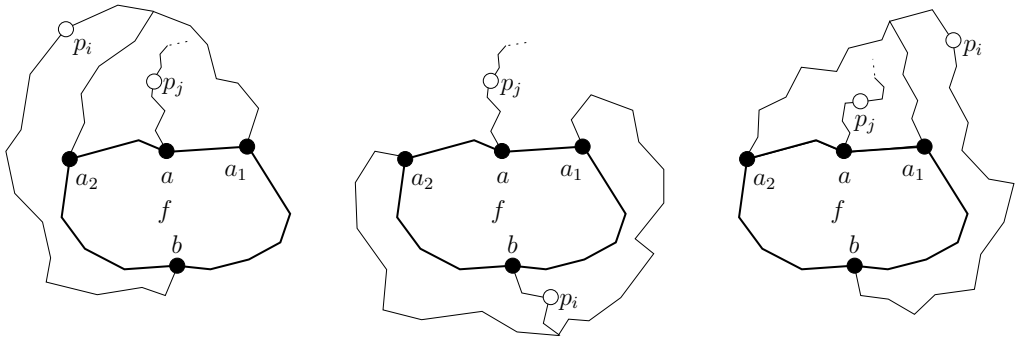


Figure 4: The situations considered in the proof of Lemma 6.

Proof. Let a' be a vertex of f_A such that either $a' <_b^f a_1$ or $a' >_b^f a_2$. Portal p_i is on some shortest path P_1 from a_1 to b. Pick P_1 such that the subpath P_1' from a_1 to p_i makes sharpest possible left turns as described in the previous section. In a similar way, we define P_2 and P_2' for a_2. Note that any path from a to b must intersect either P_1' or P_2', see Figure 4.

Portal p_j is on a shortest path P from a to b. Pick P such that the subpath P' from a to p_j makes sharpest possible left turns. Path P' cannot cross P_1 for otherwise $c_a(b) \neq c_j$, and $p_j \notin P_1'$ for otherwise $c_{a_1}(b) \neq c_i$. Similarly, P' cannot cross P_2 for otherwise $c_{a_2}(b) \neq c_i$, and $p_j \notin P_2'$ for otherwise $c_{a_2}(b) \neq c_i$. Furthermore, p_j cannot belong to the subpath shared by P_1 and P_2 from p_i to b since then $c_a(b) \neq c_j$.

It follows that any shortest path from a' to p_j crosses either P_1' or P_2'. This implies that $c_b(a') = c_{a'}(b) \neq c_j$, as requested. □

Lemma 7. *The number of dual colour intervals of a vertex $b \in f_B$ is $O(k)$.*

Proof. Let $N(k)$ be the maximum number of dual colour intervals of b when the number of distinct colours in these intervals is exactly k. We will show that $N(k) \leq 2k - 1$. The proof is by induction on $k \geq 1$. If $k = 1$ then there is only one dual colour interval and we have $N(k) = 1 = 2k - 1$.

Now, suppose that $k > 1$ and that $N(k') \leq 2k' - 1$ for all k' less than k. Let c_i be the colour of the first dual colour interval w.r.t. the order $<_b^f$. By Lemma 6, there is a finite number r of dual colour intervals with colour c_i (in fact at most k). Let $I_1, \ldots, I_s$ be the intervals between each consecutive pair of these dual colour intervals and let $k_1, \ldots, k_s$ be the number of colours in each of them. Note that $s \geq r - 1$. Also note that by the choice of c_i and by Lemma 6, two points in two different I-intervals cannot have the

same colour since for at least one of the two intervals, the colour of the dual colour interval preceding and succeeding it is c_i. From this and from the fact that the I-intervals do not contain colour c_i, $\sum_{j=1}^{s} k_j = k-1$. Applying the induction hypothesis, this gives

$$N(k) \leq r + \sum_{j=1}^{s} N(k_j) \leq r + \sum_{j=1}^{s} 2k_j - 1 = r - s + 2(k-1) \leq 1 + 2(k-1) = 2k-1.$$

□

We are now ready to bound the running time for the phase 1 algorithm.

Theorem 3. *The algorithm for phase* 1 *has* $O(|f|k)$ *running time.*

Proof. Consider an edge $e = (u, v)$ of f_A and an edge $e' = (u', v')$ of f_B such that e' is an edge of a type 1-interval of e. Let $c_i = c_u(u') = c_u(v')$ and let $c_j = c_v(u') = c_v(v')$. Since u' is a vertex of f_B, dual colours $c_{u'}(u) = c_i$ and $c_{u'}(v) = c_j$ are well-defined and since $i \neq j$ by assumption, these two dual colours are distinct, implying that two dual colour intervals of b meet at e.

It follows by the above and by Lemma 7 that for each e', the phase 1 algorithm picks $O(k)$ edges e. Thus, the total number of edge pairs considered by the algorithm is $O(|f|k)$. By our earlier discussion, this suffices to show the theorem. □

5.2 Phase 2

We now consider the problem of computing the maximum detour between points of edges of f_A and points of type 2-intervals.

For any pair (e, P), where e is an edge of f_A and P is a type 2-interval of e, there is a portal p_i such that for any point $p \in e$ and any point $q \in P$, p_i is on a shortest path from p to q. In the following, we consider all such pairs for a fixed p_i. We will show that computing the maximum detour δ_i over all these pairs can be done in $O(|f| \log^2 |f|)$ expected time. From this, it will follow that phase 2 takes $O(|f|k \log^2 |f|)$ expected time.

Before showing how to compute δ_i, we need the idea of a canonical decomposition of f_B, defined next.

5.2.1 Canonical Decomposition

Define $b_1, \ldots, b_m$ as the interval of vertices of f_B ordered according to $<_a^f$ for some arbitrary vertex $a \in f_A$. Consider splitting this interval at vertex $b_j = b_{\lceil m/2 \rceil}$ and repeat this process recursively on the two sub-intervals, stopping when an interval containing only two vertices is reached. This gives us $O(m) = O(|f|)$ intervals of total size $O(|f| \log |f|)$ which we refer to as *canonical intervals*. The subgraphs of f_B induced by these canonical intervals are referred to as *canonical subgraphs*. The set of these subgraphs, which we denote by $\mathcal{C}$, can be found in $O(|f| \log |f|)$ time.

Every sub-interval of $b_1, \ldots, b_m$ can be decomposed into $O(\log |f|)$ canonical intervals in $O(\log |f|)$ time. This is easily seen by applying a greedy algorithm that picks canonical intervals as large as possible. We refer to such a decomposition as a *canonical decomposition*.

Let e be an edge of f_A. By the assumption earlier that type 2-intervals end in vertices and by Theorem 1, the union of all type 2-intervals of e of colour c_i is exactly the set of points of f_B between two vertices of $b_1, \ldots, b_m$. We compute a canonical decomposition of the sub-interval between these two vertices and add a pointer to e from each canonical subgraph corresponding to canonical intervals in this decomposition. This is done for all e in f_A.

When finished we have a total of $O(|f| \log |f|)$ pointers from canonical subgraphs in $\mathcal{C}$ to edges of f_A. Note that some canonical subgraphs may contain pointers to several edges. The total time spent on constructing $\mathcal{C}$ and on finding pointers is $O(|f| \log |f|)$.

Observe that δ_i is the maximum of $\delta_G(e, C)$ over all pairs consisting of an edge $e \in f_A$ and a canonical subgraph $C \in \mathcal{C}$ with a pointer to e. We now show how to compute this maximum in $O(|f| \log^2 |f|)$ expected time.

5.2.2 Sweep-plane Algorithm

To efficiently compute the maximum over pairs of edges and canonical subgraphs, we will use the idea of lifting and lowering points followed by a sweep-plane algorithm as described in [8]. In order to do this, we consider the following decision problem below: given $\delta \in \mathbb{R}$, is $\delta_i \geq \delta$? If we can answer this quickly we can compute δ_i in low expected time using a randomized algorithm by Chan [4] as described in [8].

For each canonical subgraph $C \in \mathcal{C}$, we lift each point $p \in C$ to height $d_G(p_i, p)$. And for each edge $e \in f_A$, we lower each point $p \in e$ to height

$-d_G(p, p_i)$. Since we have precomputed SSSP lengths for portal p_i, this lifting/lowering can be done in $O(|f| \log |f|)$ time since the total size of all canonical subgraphs and edges is $O(|f| \log |f|)$.

Let e be a vertex in f_A and let C be a canonical subgraph with a pointer to e. Then it is clear that the height difference between a point $p \in e$ and a point $q \in C$ equals $d_G(p, q)$.

For each lifted and lowered point p, we associate a cone extending downwards from p and spanning an angle of $\alpha = 2 \arctan(1/\delta)$. Then as shown in [8], $\delta_i \geq \delta$ if and only if a cone of a lowered point of some edge is contained in a cone of a lifted point of some canonical subgraph with a pointer to that edge.

Now, we sweep a plane over the cones. The sweep-plane is parallel to the x-axis and forms an angle of $(\pi - \alpha)/2$ with the xy-plane. During the sweep, we maintain, for each canonical subgraph C, the intersection between the sweep-plane and the upper envelope of lifted points of C together with lowered points in edges that C points to. If it is detected that a cone of a lowered point is contained in a cone of a lifted point, the algorithm reports that $\delta_i \geq \delta$ and if no such event occurs, the algorithm reports that $\delta_i < \delta$. This solves our decision problem.

It follows from the results of [8] that maintaining intersections between the sweep-plane and upper envelopes takes a total of $O(|f| \log^2 |f|)$ time since the number of sweep-plane event points is $O(|f| \log |f|)$ and each event point takes $O(\log |f|)$ time to handle. However, this is under the assumption that no cone of a lifted point is contained in the interior of another cone of a lifted point and that no cone of a lowered point is contained in the interior of another cone of a lowered point (see [8] for details).

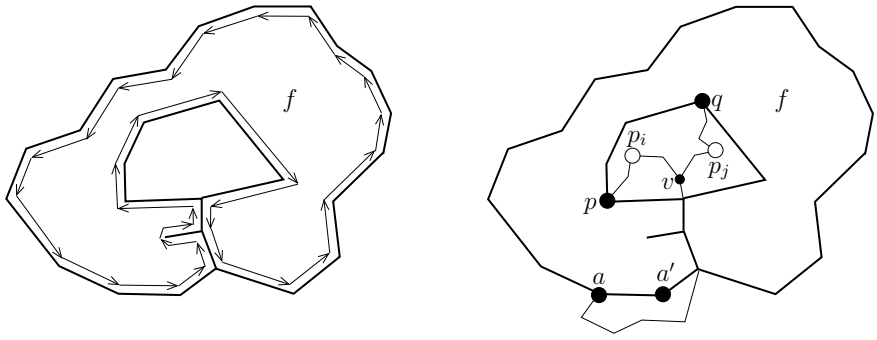
Since all lifted points belong to G_B, we may satisfy this assumption for cones of lifted points by solving our decision problem using only values $\delta \geq \delta_G(G_B)$ (a similar trick is used in [8] for trees). For suppose that $\delta \geq \delta_G(G_B)$ and consider two points p and p' belonging to the same canonical subgraph.

Let h be the height of p and let h' be the height of p'. Assume w.l.o.g. that $h \geq h'$. Then

$$h - h' = d_G(p, p_i) - d_G(p', p_i).$$

We know that $d_G(p, p') \leq \delta_G(G_B)|pp'| \leq \delta|pp'|$. This gives

$$\frac{h - h'}{|pp'|} = \frac{d_G(p, p_i) - d_G(p', p_i)}{|pp'|} \leq \frac{d_G(p, p')}{|pp'|} \leq \delta,$$



(a) (b)

Figure 5: (a): The walk of f when f is non-simple. (b): Theorem 1 holds also when f is non-simple.

showing that the cone associated with p' cannot belong to the interior of the cone associated with p. A similar argument shows that with $\delta \geq \delta_G(G_A)$, no cone of a lowered point is contained in the interior of another cone of a lowered point.

So by recursively computing $\delta_G(G_B)$ and $\delta_G(G_A)$ before computing $\delta_G(G_A, G_B)$ it follows that the above decision problem can be solved in $O(|f| \log^2 |f|)$ time and Chan's algorithm gives us the following result.

Theorem 4. *The algorithm for phase* 2 *has* $O(|f|k \log^2 |f|)$ *expected running time.*

What remains in order to compute $\delta_G(G_A, G_B)$ is to handle the case where f is non-simple. This is the topic of the next section.

5.3 Non-simple Faces

Suppose f is non-simple. The walk of f defined in the beginning of Section 4 still applies if we inflate the edges and allowing an edge to be visited twice, see Figure 5(a). Then running through the arguments of the preceding sections, it can be seen that all results still hold (as an example, compare Figure 5(b) with Figure 2).

We have now obtained the following result.

Theorem 5. $\delta_G(G_A, G_B)$ *can be computed in* $O(n^{3/2} \log^2 n)$ *expected time.*

6 Dealing with Recursive Calls

In the preceding sections, we have shown how to compute $\delta_G(G_A, G_B)$ in $O(n^{3/2}\log^2 n)$ time. In this section, we present an algorithm that recursively computes $\delta_G(G_A)$ (computing $\delta_G(G_B)$ is dealt with in a similar way) and we will analyze its running time. From this analysis it will follow that δ_G can be computed in $O(n^{3/2}\log^3 n)$ expected time.

Let $n_A = |A|$ be the number of vertices of A. We start by applying the separator theorem to G_A which partitions A into three sets, A_1, A_2, and C_A such that no edge joins a vertex in A_1 with a vertex in A_2, neither A_1 nor A_2 contains more than $n_A/2$ vertices, and C_A contains no more than $\frac{2\sqrt{2}}{1-\sqrt{2/3}}\sqrt{n_A}$ vertices. Let G_{A_1} be the subgraph of G_A induced by $A_1 \cup C_A$ and let G_{A_2} be the subgraph of G_A induced by $A_2 \cup C_A$. We have

$$\delta_G(G_A) = \max\{\delta_G(G_{A_1}), \delta_G(G_{A_2}), \delta_G(G_{A_1}, G_{A_2})\}.$$

In the following, we will present an algorithm for computing $\delta_G(G_{A_1}, G_{A_2})$. From this it will easily follow how to handle subgraphs in any recursion level.

Let P be the set of portals for G and let $P_A = P \cup C_A$. Given a vertex $a_1 \in A_1$ and a vertex $a_2 \in A_2$, any path in G from a_1 to a_2 must contain at least one vertex of P_A. Thus, by defining P_A as the set of portals for G_A, most of the results we presented in order to compute $\delta_G(G_A, G_B)$ now also apply to the problem of computing $\delta_G(G_{A_1}, G_{A_2})$. However, three problems need to be dealt with.

The first problem is that G_A may be disconnected and it causes problems when defining cyclic orderings of faces. We deal with this as follows. Let f be a face of G such that $f \cap A \neq \emptyset$. Then as the cyclic ordering of the vertices of $f \cap A$, we use that which is induced by the cyclic ordering of vertices of f. Essentially, this defines faces of G_A where "holes" are allowed.

The second problem is that of computing SSSP distances from each portal of P_A to all vertices in G_A. The running time for this should depend on the size of G_A, not the size of G. To obtain this, note that we only need to compute SSSP distances from portals of C_A since we have computed distances from each portal in P to all vertices of G and in particular to vertices of G_A.

So consider a portal $p_i \in C_A$. We compute SSSP distances from p_i to vertices in G_A using Dijkstra's algorithm but with a different initialization. We are already given distances from p_i to portals in P so we add these portals to the priority queue and associate a shortest path distance from p_i to each of

them. We also add p_i to the priority queue and set the shortest distance from p_i to itself equal to zero. The rest of the algorithm runs like normal Dijkstra on G_A. This way, we find SSSP distances in G from p_i to all vertices in G_A in $O(n_A \log n_A)$ time. Since $|C_A| = O(\sqrt{n_A})$, computing SSSP distances for all portals of C_A takes a total of $O(n_A^{3/2} \log n_A)$ time.

The third and final problem we need to deal with is determining the order of colour intervals of $f \cap G_{A_2}$ for each vertex of $f \cap G_{A_1}$ where f is a face. Lemma 4 states that the order of colour intervals of a vertex of f_A can be computed in $O(k \log^2 n)$ time with $O(kn \log n)$ time for preprocessing. Looking at the proof of the lemma, we see that the preprocessing step only considers portals of G, not the set of portals P_A for G_A. It thus needs to be recomputed for G_A. But a priori, this requires that we look at the entire graph G since shortest paths from vertices in G_A to portals in P_A need not be fully contained in G_A. This will make our recursive algorithm too slow.

We modify the preprocessing step for G_A as follows. We compute π-pointers for each vertex of G_A and each portal of P_A as in the proof of Lemma 4 except that when finding a path to a portal $p_i \in P_A$ we stop if we reach a portal of P (or as before, if we reach a red edge or p_i).

This modification gives $O(k_A n_A \log n_A)$ preprocessing time where $k_A = |P_A|$. We now show how this preprocessing can be used to efficiently determine whether $p_i \prec_a^f p_j$ for two distinct portals $p_i, p_j \in P_A$ where a is a vertex of G_{A_1} belonging to a face f of G.

Let a' be the successor of a in f w.r.t. $<_a^f$. To determine whether $p_i \prec_a^f p_j$, we apply binary searches as before in paths $\overleftarrow{P_i}(a', a)$ and $\overleftarrow{P_j}(a', a)$. If a split is detected, or if it is detected that $p_i \in \overleftarrow{P_j}(a', a)$ or $p_j \in \overleftarrow{P_i}(a', a)$ then we can correctly decide if $p_i \prec_a^f p_j$.

The only problem that may arise is if both binary searches end in an edge e from which there are no π-pointers and it still has not been determined whether $p_i \prec_a^f p_j$. But in this case, one of the endpoints of e must be a portal of P_A and this portal must belong to both $\overleftarrow{P_i}(a', a)$ and $\overleftarrow{P_j}(a', a)$ implying that no colour intervals of a will get colour c_i or c_j. Hence, it is irrelevant whether $p_i \prec_a^f p_j$ and we may simply delete the two portals in the algorithm that sorts portals. When the sorting algorithm terminates, the order of the remaining portals will give the order of colour intervals of a.

With the above three modifications, we get an algorithm that computes $\delta_G(G_{A_1}, G_{A_2})$ in $O(k_A n_A \log^2 n_A)$ expected time. We compute $\delta_G(G_{A_1})$ and $\delta_G(G_{A_2})$ recursively where the portals for G_{A_1} are defined as those that be-

long to $P_A \cap G_{A_1}$ together with those obtained when applying the separator theorem to G_{A_1} (and similarly for G_{A_2}).

More generally, consider a subgraph G' in some node of the recursion tree. If the size of G' is less than some constant, a brute-force algorithm that computes APSP-distances for vertices in G' is applied to find $\delta_G(G')$ in constant time.

Otherwise, let G_{c_1} and G_{c_2} be the subgraphs in the two child nodes obtained by applying the separator theorem to G'. Let P' be the set of portals of G'. Then for $i = 1, 2$, the set of portals of G_{c_i} is defined as the union of $P' \cap G_{c_i}$ and the set of portals obtained by applying the separator theorem to G_{c_i}.

Let k_p denote the number of portals of G' that are also portals of the subgraph of G belonging to the parent node of the node containing G', let k' be the number of additional portals of G', and let n' be the number of vertices of G'. Then similar arguments as above show that $\delta_G(G_{c_1}, G_{c_2})$ can be computed in $O((k_p + k')n' \log^2 n')$ expected time. Since $k' = O(\sqrt{n'})$, this can be rewritten as $O(k_p n' \log^2 n' + n'\sqrt{n'} \log^2 n')$.

Clearly, the sum of $O(n'\sqrt{n'} \log^2 n')$ over all non-leaf nodes of the recursion tree is $O(n^{3/2} \log^2 n)$. Let us bound the sum of $O(k_p n' \log^2 n')$ over all these nodes. Let $T(n', k_p)$ denote the total time spent in the subtree of the recursion tree rooted at a node containing a graph with n' vertices and sharing k_p portals with the subgraph in the parent node. Then

$$T(n', k_p) \leq T(n'/2, k_1 + c\sqrt{n'}) + T(n'/2, k_2 + c\sqrt{n'}) + O(k_p n' \log^2 n'),$$

where $c = \frac{2\sqrt{2}}{1-\sqrt{2/3}}$ and $k_1 + k_2 \leq k_p$.

It follows that the sum of all k_p in the ith level of the recursion tree is $O(\sum_{j=0}^{i} 2^j \sqrt{n/2^j})$ so the total time spent in this level is

$$O(\sum_{j=0}^{i} 2^j \sqrt{n/2^j} n/2^i \log^2 n) = O(n^{3/2} \log^2 n \sum_{j=0}^{i} 2^{j/2-i}).$$

Since $\sum_{j=0}^{i} 2^{j/2-i} \leq \sum_{j=0}^{i} 2^{j-i} < \sum_{j=0}^{\infty} 2^{-j} = 2$ and since there are $O(\log n)$ recursion levels, we have now obtained the main result of this paper.

Theorem 6. *The maximum detour of a plane graph with n vertices can be computed in $O(n^{3/2} \log^3 n)$ expected time.*

7 Graphs with Bounded Treewidth

We now consider the problem of computing the maximum detour of a plane graph with bounded treewidth. The treewidth of a graph is, in a sense, a measure of the complexity of graph. The following definition and lemma are taken from [3].

Definition 1. *A* tree decomposition *of a graph $G = (V, E)$ is a pair (X, T), where $X = \{X_i \subseteq V | i \in I\}$ is a collection of subsets of V (called* bags*), and a tree $T = (I, F)$ with a node set I such that*

1. *$V = \cup_{i \in I} X_i$*
2. *For every edge $(u, v) \in E$ there is some bag $X_i \in X$ such that $u, v \in X_i$*
3. *For all $u \in V$, the nodes $\{i \in I | u \in X_i\}$ form a connected subtree of T*

The width *of a tree decomposition $(\{X_i | i \in I\}, T)$ is $\max_{i \in I} |X_i| - 1$. The* treewidth *of G is the minimum width over all tree decompositions of G.*

Lemma 8. *Let $w \geq 1$ be a constant. Given a graph $G = (V, E)$ with $n > w+1$ vertices and treewidth at most w, we can find in linear time a partition of V into three subsets A, B, and P such that*

1. *no edge joins a vertex in A with a vertex in B,*
2. *A and B each have between $\frac{n}{w+1} - w$ and $\frac{nw}{w+1}$ vertices,*
3. *P contains no more than w vertices, and*
4. *adding edges between the vertices of P does not change the treewidth.*

If we apply Lemma 8 to G instead of the separator theorem by Lipton and Tarjan we get the subgraph G_A of G induced by $A \cup P$ and the subgraph G_B of G induced by $B \cup P$. Since the number of portals is at most w, we may compute $\delta_G(G_A, G_B)$ in $O(wn \log^2 n) = O(n \log^2 n)$ time.

To recursively compute $\delta_G(G_A)$, we define graph G' as the graph obtained by removing all edges of G not belonging to G_A and adding an edge between each pair of portals of G. The cost of each such edge is set to the distance in G between the corresponding pair of portals. Note that the number of edges added is constant.

When we regard G' as a point set, we disregard edges added between portals. Observe that for each pair of points p and q in G', $d_{G'}(p,q) = d_G(p,q)$.

Since G' has treewidth at most w by Lemma 8, an inductive argument shows that $\delta_G(G_A) = \delta_{G'}(G_A)$ can be computed in $O(n \log^3 n)$ expected time. A similar argument shows that $\delta_G(G_B)$ can be computed in $O(n \log^3 n)$ expected time. This shows the second result of our paper.

Theorem 7. *The maximum detour of a plane graph with n vertices and bounded treewidth can be computed in $O(n \log^3 n)$ expected time.*

8 Concluding Remarks

In this paper, we showed how to compute the maximum detour of a plane graph in $O(n^{3/2} \log^3 n)$ expected time. This is an improvement over the best known algorithm with $\Theta(n^2)$ running time. We also showed that if the graph has bounded treewidth, its maximum detour can be computed in $O(n \log^3 n)$ expected time.

We believe that by using parametric search as described in [1], we can obtain an algorithm computing the maximum detour of a plane graph in $O(n^{3/2} \operatorname{polylog} n)$ *worst-case* time and in $O(n \operatorname{polylog} n)$ *worst-case* time when the graph has bounded treewidth.

It would be interesting to try to beat the quadratic time bound also for the problem of computing the stretch factor of a planar (or plane) geometric graph. This problem appears harder since pairs of vertices achieving the maximum detour need not be co-visible.

References

[1] P. K. Agarwal, R. Klein, C. Knauer, S. Langerman, P. Morin, M. Sharir, and M. Soss. Computing the Detour and Spanning Ratio of Paths, Trees and Cycles in 2D and 3D. Discrete and Computational Geometry, 39 (1): 17–37 (2008).

[2] S. Arya, G. Das, D. M. Mount, J. S. Salowe, and M. Smid. Euclidean spanners: short, thin, and lanky. Proc. 27th ACM STOC, 1995, pp. 489–498.

[3] S. Caballo and C. Knauer. Algorithms for Graphs With Bounded Treewidth Via Orthogonal Range Searching. Manuscript, Berlin, 2007.

[4] T. M. Chan. Geometric applications of a randomized optimization technique. Discrete Comput. Geom., 22(4):547–567, 1999.

[5] A. Ebbers-Baumann, R. Klein, E. Langetepe, and A. Lingas. A Fast Algorithm for Approximating the Detour of a Polygonal Chain. Comput. Geom. Theory Appl., 27: 123–134, 2004.

[6] D. Eppstein. Spanning trees and spanners. In J.-R. Sack and J. Urrutia, editors, Handbook of Computational Geometry, pages 425–461, Elsevier Science Publishers, Amsterdam, 2000.

[7] G. N. Frederickson Fast algorithms for shortest paths in planar graphs, with applications. SIAM J. Comput., 16 (1987), pp. 1004–1022.

[8] S. Langerman, P. Morin, and M. Soss. Computing the Maximum Detour and Spanning Ratio of Planar Paths, Trees and Cycles. In Proc. 19th International Symposium on Theoretical Aspects of Computer Science, volume 2285 of Lecture Notes in Computer Science, pages 250–261, Springer-Verlag, 2002.

[9] X. Y. Li and Y. Wang. Efficient construction of low weighted bounded degree planar spanner. Int. J. Comput. Geometry Appl. 14(1–2):69–84 (2004).

[10] R. J. Lipton and R. E. Tarjan. A Separator Theorem for Planar Graphs. STAN-CS-77-627, October 1977.

[11] G. Narasimhan and M. Smid. Geometric Spanner Networks. Cambridge University Press, 2007.

[12] M. Smid. Closest point problems in computational geometry. In J.-R. Sack and J. Urrutia, editors, Handbook of Computational Geometry, pages 877–935, Elsevier Science Publishers, Amsterdam, 2000.